# Series de Fourier

Angel Octavio Parada Flores

12 de Febrero de 2021

## Desarrollo del código

Se importan las librerías necesarias

```
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
import math as mt
```

Debido al uso de la librería `sympy` es necesario definir la variable $x$

```
x = sp.symbols("x")
```

En seguida, se sabe que los coeficientes de Fourier se cálculan por medio de

$$B_0 = \frac{1}{\lambda_1}\int_{z_1}^{z_1+\lambda_1} F(z)dz,$$
$$A_m = \frac{2}{\lambda_1}\int_{z_1}^{z_1+\lambda_1} F(z)\sin{(mk_1z)}dz,$$
$$B_m = \frac{2}{\lambda_1}\int_{z_1}^{z_1+\lambda_1} F(z)\cos{(mk_1z)}dz,$$

con $k_1 = \frac{2\pi}{\lambda_1}$. Por lo que es necesario, primero, definir las variables $F(z)$ (usando la variable $x$ en lugar de $z$), $\lambda_1$ (periodo), $z_1$ (valor inicial en el que se define la función) y $m$ (número de armónicos). Dado que se busca que el programa sea lo más interactivo posible, se definen como variables de entrada

```
m = int(input('Número de armónicos:', ))
I = float(input('Extremo inicial del periodo:',))
T = float(input('Periodo:', ))
F = sp.sympify(input('Función:', ))
```

Con las anteriores variables, se puede definir

```
k = mt.pi/T
```

Se pretende hacer la serie de Fourier para los siguientes tipos de funciones:

1. $$F(z), \qquad z \in [z_1, z_1 + \lambda_1]$$

2. $$F(z) = \begin{cases} g(z), & z \in [z_1, z_1 + \frac{\lambda_1}{2}] \\ -g(z), & z \in [z_1 + \frac{\lambda_1}{2}, z_1 + \lambda_1] \end{cases}$$

3. $$F(z) = \begin{cases} g(z), & z \in [z_1, z_1 + \frac{\lambda_1}{2}] \\ h(z), & z \in [z_1 + \frac{\lambda_1}{2}, z_1 + \lambda_1] \end{cases}$$

Para estos propósitos, se introducen dos variables que determinarán la forma de la función que se tendrá

```
p1 = input('¿Se va a reflejar la función? [S/n]:',)
p2 = 0
if p1 == 'n':
    p2 = input('¿Es una función en dos partes? [S/n]:',)

G = 0
if p1 == 'S':
    G = -F.subs(x, x - T/2)
elif p1 == 'n' and p2 == 'n':
    G = F
elif p1 == 'n' and p2 == 'S':
    G = sp.sympify(input('Función 2:', ))
```

Así, el segundo `if` corresponderá a una función del tipo 2, el primer `elif` del segundo `if` a una función del tipo 1, y el segundo `elif` del segundo `if` a una función del tipo 3.

En seguida, con base en las formulas de los coeficientes $B_0$, $A_m$ y $B_m$ (y aprovechando las utilidades de la libreria `sympy`), se calculan los coeficientes para la serie de Fourier:

```
B_0 = sp.integrate(F/T, (x, I, I+T/2)) + sp.integrate(G/T, (x, I+T/2, I+T))
print('B_0=', B_0)
A_m = []
B_m = []
for i in range (1, m+1):
    A_m.append(sp.integrate(F*(2/T)*sp.sin(i*2*k*x), (x, I, I+T/2))
    + sp.integrate(G*(2/T)*sp.sin(i*2*k*x), (x, I+T/2, I+T)))
    B_m.append(sp.integrate(F*(2/T)*sp.cos(i*2*k*x), (x, I, I+T/2))
    + sp.integrate(G*(2/T)*sp.cos(i*2*k*x), (x, I+T/2, I+T)))
```

Con el objetivo de calcular los coeficientes $A$ y $B$ para cada $m$, se definió un vector vacio que fue llenandose con un ciclo `for`. Ahora, la serie de Fourier se define

$$F(z) = B_0 + \sum_{m=1}^{\infty} A_m \sin(mk_1 z) + \sum_{m=1}^{\infty} B_m \cos(mk_1 z).$$

Por lo que, para algún valor de $z$, debe calcularse la suma sobre $m$ del producto de los coeficientes con su respectiva función trigonométrica, es decir:

```
def Fourier(B_0, A_m, B_m, m, k, y):
    VA = 0
    VB = 0
    for i in range(0, m):
        VA += A_m[i]*np.sin((i+1)*2*k*y)
        VB += B_m[i]*np.cos((i+1)*2*k*y)
    return B_0 + VA + VB
```

Así, añadiendo comandos para gráficar de `matplotlib`, el programa quedaría de la siguiente manera:

```
#Autor: Angel Octavio Parada Flores
import matplotlib.pyplot as plt
import numpy as np
import sympy as sp
import math as mt

def Fourier(B_0, A_m, B_m, m, k, y):
    VA = 0
    VB = 0
    for i in range(0, m):
        VA += A_m[i]*np.sin((i+1)*2*k*y)
        VB += B_m[i]*np.cos((i+1)*2*k*y)
    return B_0 + VA + VB

x = sp.symbols("x")
m = int(input('Número de armónicos:', ))
I = float(input('Extremo inicial del periodo:',))
T = float(input('Periodo:', ))
F = sp.sympify(input('Función:', ))
p1 = input('¿Se va a reflejar la función? [S/n]:',)
p2 = 0
if p1 == 'n':
    p2 = input('¿Es una función en dos partes? [S/n]:',)

G = 0
if p1 == 'S':
    G = -F.subs(x, x - T/2)
elif p1 == 'n' and p2 == 'n':
    G = F
elif p1 == 'n' and p2 == 'S':
    G = sp.sympify(input('Función 2:', ))

k = mt.pi/T
```

```
B_0 = sp.integrate(F/T, (x, I, I+T/2)) + sp.integrate(G/T, (x, I+T/2, I+T))
A_m = []
B_m = []
for i in range (1, m+1):
    A_m.append(sp.integrate(F*(2/T)*sp.sin(i*2*k*x), (x, I, I+T/2))
    + sp.integrate(G*(2/T)*sp.sin(i*2*k*x), (x, I+T/2, I+T)))
    B_m.append(sp.integrate(F*(2/T)*sp.cos(i*2*k*x), (x, I, I+T/2))
    + sp.integrate(G*(2/T)*sp.cos(i*2*k*x), (x, I+T/2, I+T)))

y = np.arange(-T*3, T*3, 0.01)
plt.figure(figsize=(7,7))
plt.plot(y, Fourier(B_0, A_m, B_m, m, k, y), label = 'Núm. armónicos=' + repr(m))
plt.legend()
plt.grid()
plt.show()
```

# Resultados

## Función triángulo

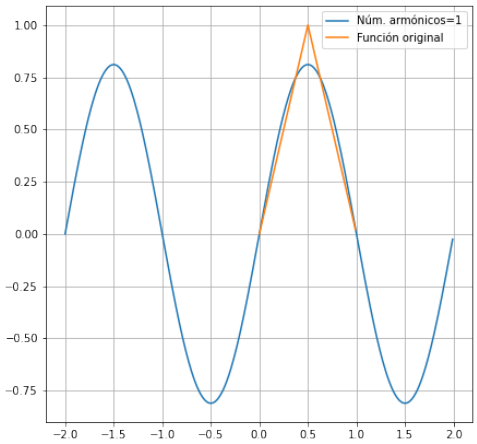
Como prueba se uso la función

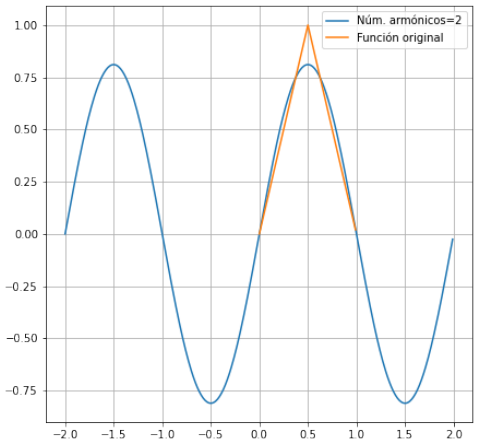
$$-2|x - \frac{1}{2}| + 1, \quad x \in [0, 1].$$

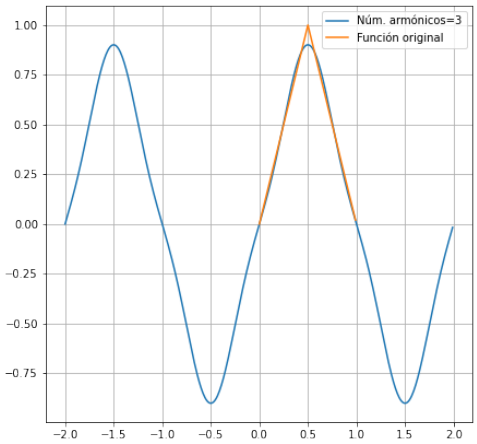
Se obtuvieron los siguientes coeficientes:

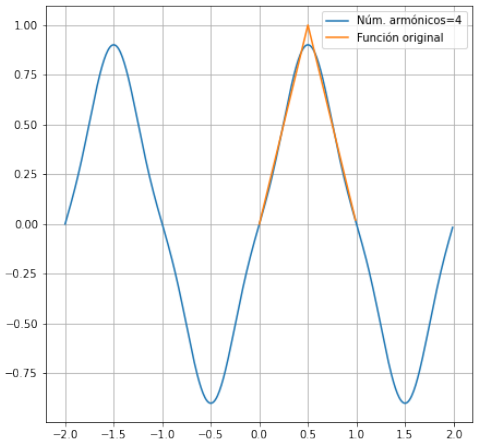
- $B_0 = 0, B_1 = -1.110 \times 10^{-16}, B_2 = 2.776 \times 10^{-17}, B_3 = 0, B_4 = 0$
- $A_1 = 0.811, A_2 = 5.551 \times 10^{-17}, A_3 = -0.090, A_4 = -2.776 \times 10^{-17}$

Gráficando:


Núm. armónicos=1
Función original
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Núm. armónicos=2
Función original
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0




## Función triángulo desplazada

Como prueba se uso la función

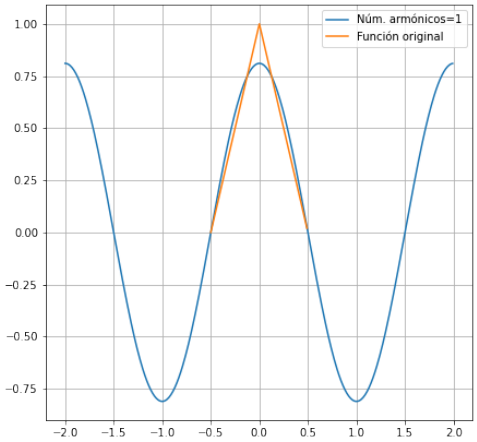
$$-2|x| + 1, \quad x \in [-\frac{1}{2}, \frac{1}{2}].$$

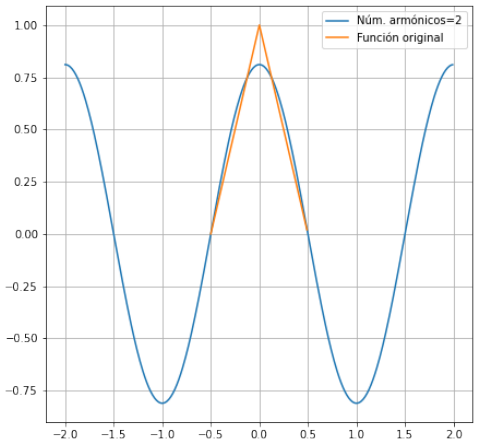
Se obtuvieron los siguientes coeficientes:

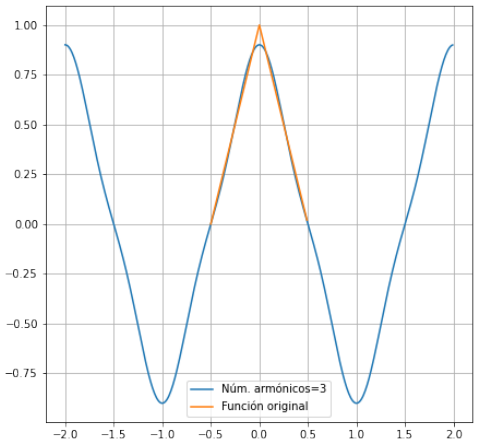
- $B_0 = 0, B_1 = 0.811, B_2 = 0, B_3 = 0.090, B_4 = 1.735 \times 10^{-18}$

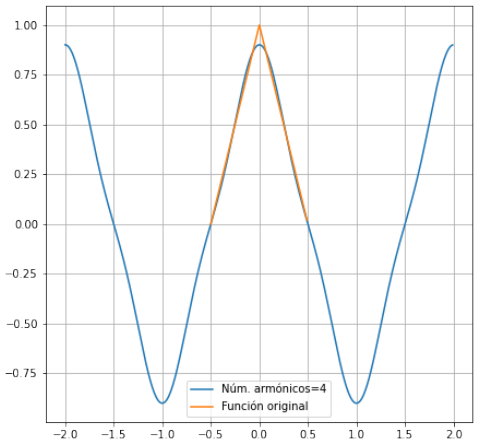
- $A_1 = -5.551 \times 10^{-17}, A_2 = 2.776 \times 10^{-17}, A_3 = 0, A_4 = 1.388 \times 10^{-17}$

Gráficando:




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Núm. armónicos=3
Función original


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Núm. armónicos=4
Función original